Kapitola 1

Moderní programování v JavaScriptu

Jazyk JavaScript se vyvíjel postupně, ale trvale. Za minulé desetiletí se z jednoduchého programovacího jazyka na hraní stal respektovaný programovací jazyk, který používají společnosti a vývojáři na celém světě k vytváření neuvěřitelných aplikací. Moderní programovací jazyk JavaScript je solidní, robustní a neuvěřitelně mocný nástroj. Na většině toho, čemu se věnuje tato kniha, můžeme vidět, co činí moderní aplikace v JavaScriptu tak odlišné. Většina myšlenek v této kapitole nevznikla jen tak z ničeho nic, ale utvářely je postupně tisíce inteligentních programátorů, myšlenky se postupně vytříbily až do podoby, v jaké je známe dnes. Takže ať to dále neoddalujeme, pojďme se podívat na moderní programovaní v JavaScriptu.

Objektově orientovaný JavaScript

Z pohledu jazyka není vůbec nic moderního na objektově orientovaném programování nebo objektově orientovaném JavaScriptu. JavaScript byl od počátku navržen jako objektově orientovaný jazyk. Avšak jak se jazyk JavaScript postupně vyvíjel, programátoři jiných programovacích jazyků (jako Ruby, Python a Perl) začali do jazyka prosazovat své programátorské názory.

Objektově orientovaný kód jazyka JavaScript vypadá a chová se jinak než v ostatních objektových jazycích. Proč je tento kód tak jedinečný, uvidíme v kapitole 2, ale nyní se podívejme na jednoduchý příklad, ať vidíme, jak se píše moderní kód v JavaScriptu. Příklad na výpise 1.1 obsahuje dva konstruktory, ukazuje jednoduché párování objektů, které lze použít pro předměty ve škole.

Výpis 1.1. Reprezentace předmětů a rozvrhů v JavaScriptu

```
// Konstruktor pro objekty typu Predmet.
// Parametry jsou dva textové řetězce, nazev a učitel.
function Predmet(nazev, ucitel) {
  // Uložíme je do lokálních atributů objektu.
  this.nazev = nazev;
  this.ucitel = učitel;
}

// Metoda třídy Predmet generuje textový řetězec, který
// můžeme použít pro zobrazení informací o předmětu.
```

```
Predmet.prototype.zobraz = function() {
  return this.ucitel + " vyučuje " + this.nazev;
};

// Konstruktor objektů typu Rozvrh, kterému předáme
// na vstupu pole předmětů.
function Rozvrh(predmety) {
  this.predmety = predmety;
}

// Metoda vytvoří textový řetězec reprezentující rozvrh předmětů.
Rozvrh.prototype.zobraz = function() {
  var str = "";

  // Procházíme všemi předměty a vytváříme textový řetězec s informacemi.
  for (var i=0; i < this.predmety.length; i++)
    str += this.predmety[i].zobraz() + " ";

  return str;
};
```

Jak vidíme na výpise 1.1, daný příklad obsahuje většinu základních objektově orientovaných prvků, ale jejich struktura se mírně liší od běžných objektově orientovaných jazyků. Můžeme vytvářet konstruktory objektů a metody, přistupovat nebo získávat vlastnosti objektu. Příklad užití těchto dvou tříd v programu vidíme na výpise 1.2.

Výpis 1.2. Tvorba vlastního rozvrhu

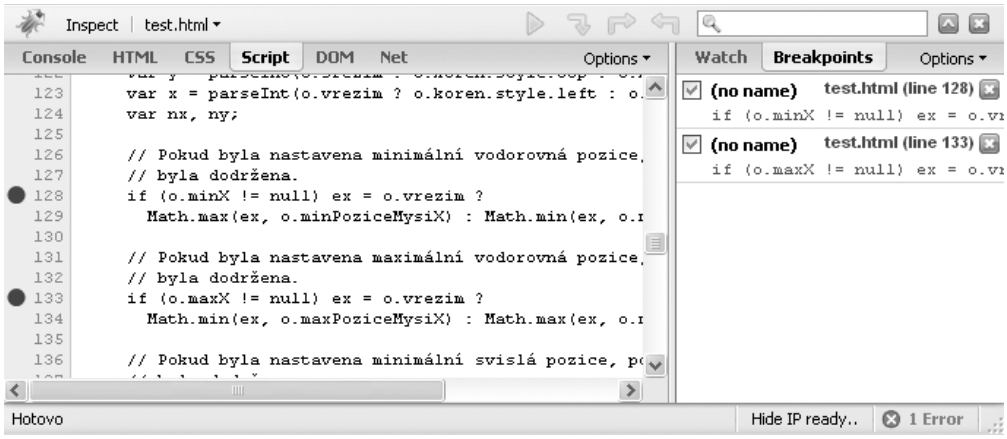
```
// Vytvoříme nový objekt typu Rozvrh a uložíme jej do
// proměnné 'mujRozvrh'.
var mujRozvrh = new Rozvrh([
  // Vytvoříme pole objektů typu Predmet, které je jediným
  // parametrem konstruktoru objektů typu Rozvrh.
  new Predmet( "Tělocvik", "Novák" ),
  new Predmet( "Matematika", "Hliněný" ),
  new Predmet( "AJ", "Procházková" )
]);

// Zobrazíme informace o rozvrhu ve výstražném okně.
alert( mujRozvrh.display() );
```

Jak byl JavaScript přijímán programátory jako běžný programovací jazyk, stával se objektově orientovaný kód více oblíbený. Tato kniha obsahuje různé části objektově orientovaného kódu v JavaScriptu, které názorně ukazují, jak objektově navrhovat a implementovat kód.

Testování zdrojového kódu

Kromě psaní objektově orientovaného kódu je nutné k vytváření profesionálních aplikací v JavaScriptu použít nějaké robustní prostředí pro testování kódu. Provádět testování kódu je nezbytné zejména tehdy, když vytváříte kód, který budou aktivně používat nebo udržovat další programátoři.

V kapitole 4 najdeme několik nástrojů, které lze použít pro vývoj testovacího systému, spolu s jednoduchým laděním složitých aplikací. Jedním z těchto nástrojů je zásuvný modul Firebug pro prohlížeč Firefox. Firebug poskytuje řadu užitečných nástrojů, jako například chybovou konzoly, zaznamenávání žádostí HTTP, ladění a inspekce prvků. Obrázek 1.1 ukazuje, jak probíhá ladění kódu v zásuvném modulu Firebug.



Obrázek 1.1. Ukázka zásuvného modulu Firebug v akci

Důraz na tvorbu čistého kódu, který lze snadno testovat, není přehnaný. Jakmile začneme vytvářet nějaký čistě objektově orientovaný kód a připojíme k tomu testování, určitě budeme muset tomuto tvrzení dát za pravdu.

Rozčleňování do balíků pro distribuci

Posledním krokem k vytváření moderního profesionálního kódu v JavaScriptu je proces rozčlenění do balíků pro distribuci. Jak programátoři začali používat stále více kódu v JavaScriptu na svých stránkách, pravděpodobnost vzniku konfliktů vzrůstala. Jestliže mají dvě knihovny v JavaScriptu shodně pojmenovanou proměnnou, třeba `data`, nebo se oba rozhodnou přidat událost, avšak každý rozdílným způsobem, může dojít ke konfliktu a vynoří se nepříjemné chyby.

Vrcholem úspěchu při vývoji knihovny v JavaScriptu je schopnost jednoduše vypustit ukazatel `<script>` na ni, a přesto neovlivnit její funkci. Existuje nespočet technik a řešení, které používají programátoři, aby udrželi svůj kód čistý a univerzálně kompatibilní.

Nejoblíbenější technikou pro ochranu kódu před ovlivněním jiným kódem v JavaScriptu je použít prostory jmen. Dobrým příkladem (ne však nejlepším nebo nejužitečnějším) toho, jak to funguje v praxi, je knihovna veřejného uživatelského rozhraní, kterou vyvinula společnost Yahoo a je volně dostupná každému. Příklad užití této knihovny vidíme na výpise 1.3.

Výpis 1.3. Přidání události k prvku pomocí knihovny uživatelského rozhraní Yahoo

```
// Přidáme obsluhu události přejetí kurzorem myši nad prvkem
// s identifikátor 'body'
YAHOO.util.Event.addListener('body','mouseover',function(){

  // a změníme barvu pozadí prvku na červenou.
  this.style.backgroundColor = 'red';

});
```

Problémem s prostory jmen je, že neexistuje souvislost mezi knihovnami, abychom mohli budovat strukturu knihoven. Pro tento případ jsou centrální repozitáře zdrojových kódů, jako například JSAN (JavaScript Archive Network), velmi užitečné. JSAN obsahuje stálý soubor pravidel pro strukturování knihoven spolu se způsobem, jak rychle a snadno importovat další knihovny, na kterých náš kód závisí. Obrázek 1.2 ukazuje, jak vypadá hlavní distribuční centrum JSAN.

Obtíže při vývoji čistého kódu, který lze dělit do balíků, popisuje kapitola 3. Další úskalí této činnosti, jako například kolize obsluhy událostí, lze najít v kapitole 6.

Obrázek 1.2. Veřejný repozitář zdrojových kódů JSAN

Nevtíravé skriptování DOM

Vytvořit jádro pro čistý, snadno testovatelný zdrojový kód a kompatibilní distribuce je úkolem tzv. nevtíravého (unobtrusive) skriptování DOM. Psát nevtíravý zdrojový kód znamená úplné oddělení obsahu HTML (dat ze serveru) a kódu v JavaScriptu, který činí tento obsah dynamickým. Podstatným vedlejším efektem tohoto úplného oddělení je, že se náš kód umí přizpůsobit webovému prohlížeči. Toho lze využít tak, že nabídneme pokročilý obsah prohlížečům, které jej podporují, a stále budou moci běžný obsah zobrazit prohlížeče, které tento pokročilý obsah nepodporují.

Psaní moderního nevtíravého kódu má dvě stránky – Document Object Model (DOM) a události JavaScriptu. Tato kniha detailně popíše oba aspekty.

Document Object Model

DOM je oblíbenou formou reprezentace dokumentů XML. Nejedná se o nejrychlejší nebo nejjednodušší způsob, ale je nejběžnější, DOM je implementován ve spoustě programovacích jazyků (jako například Java, Perl, PHP, Ruby, Python a JavaScript). DOM byl vytvořen, aby poskytl intuitivní způsob, jak procházet hierarchii XML.

Protože validní HTML je podmnožinou XML, existence efektivního způsobu pro analýzu a prohlížení dokumentů DOM je nezbytná, aby bylo programování v JavaScriptu jednodušší. Uvědomme si, že velká část interakcí v JavaScriptu probíhá mezi JavaScriptem a různými prvky jazyka HTML, které obsahuje nějaká webová stránka, proto je DOM vhodný nástroj, který tento proces zjednodušuje. Příklady použití DOM k procházení, vyhledání a zpracování různých prvků uvnitř webové stránky můžeme vidět na výpise 1.4.

Výpis 1.4. Lokalizace a zpracování různých DOM prvků pomocí Document Object Model

```
<html>
<head>
  <title>Úvod do DOM</title>
  <script>
    // Nemůžeme zpracovat DOM, dokud není dokument
    // zcela načten.
    window.onload = function(){

      // Najdeme všechny prvky <li> v dokumentu
      var li = document.getElementsByTagName("li");

      // a přidáme jim okraj.
      for ( var j = 0; j < li.length; j++ ) {
        li[j].style.border = "1px solid #000";
      }

      // Vyhledáme prvky s identifikátorem 'kdekoliv'
      var vsude = document.getElementById( "kdekoliv" );

      // a odstraníme je z dokumentu.
```

```
      vsude.parentNode.removeChild( vsude );
    };
  </script>
</head>
<body>
  <h1>Úvod do DOM</h1>
  <p class="test">Existuje celá řada důvodů, proč je DOM úžasný,
    zde jich několik máme:</p>
  <ul>
    <li id="kdekoliv">Lze na něj narazit všude.</li>
    <li class="test">Snadno se používá.</li>
    <li class="test">Pomůže nám najít to, co potřebujeme,
      opravdu rychle.</li>
  </ul>
</body>
</html>
```

DOM je prvním krokem k psaní nevtíravého zdrojového kódu v JavaScriptu. Jelikož jsme schopni rychle a jednoduše prohledat dokument HTML, všechny výsledné interakce mezi JavaSciptem a HTML budou mnohem jednodušší.

Události

Události jsou lepidlem, které drží pohromadě všechny interakce uvnitř aplikace. V dobře navržené aplikaci v JavaScriptu budeme mít zdroje dat a jejich vizuální reprezentaci (v HTML DOM). Abychom synchronizovali tyto dva aspekty, musíme sledovat akce uživatelů a přizpůsobovat tomu uživatelské rozhraní. Kombinace DOM a událostí JavaScriptu činí moderní webové aplikace tím, čím jsou.

Všechny moderní webové prohlížeče poskytují řadu událostí, které spustí, kdykoliv nastane příslušná akce, jako například pohyb myší, stisknutí klávesy na klávesnici nebo zavření webové stránky. Díky těmto událostem můžeme psát kód, který bude proveden, když daná událost nastane. Příklad této interakce zobrazuje výpis 1.5, kde měníme pozadí prvku `<li>`, když nad ním uživatel přejede kurzorem myši.

Výpis 1.5. Vizuální efekty pomocí DOM a událostí

```
<html>
<head>
  <title>Úvod do DOM</title>
  <script>
    // Nemůžeme zpracovat DOM, dokud není dokument
    // zcela načten.
    window.onload = function(){

      // Najdeme všechny prvky <li>, abychom k nim připojili
      // obsluhy událostí.
      var li = document.getElementsByTagName("li");
      for ( var i = 0; i < li.length; i++ ) {

        // Připojíme k prvku <li> obsluhu událostí,
```

```
        // která změní pozadí prvku <li> na modrou barvu.
        li[i].onmouseover = function() {
          this.style.backgroundColor = 'blue';
        };

        // Připojíme k prvku <li> obsluhu událostí mouseout,
        // která změní pozadí prvku <li> zpět na původní bílou barvu.
        li[i].onmouseout = function() {
          this.style.backgroundColor = 'white';
        };

      }

    };
  </script>
</head>
<body>
  <h1>Úvod do DOM</h1>
  <p class="test">Existuje celá řada důvodů, proč je DOM úžasný,
    zde jich několik máme:</p>
  <ul>
    <li id="kdekoliv">Lze na něj narazit všude.</li>
    <li class="test">Snadno se používá.</li>
    <li class="test">Pomůže nám najít to, co potřebujeme,
      opravdu rychle.</li>
  </ul>
</body>
</html>
```

Události JavaScriptu jsou komplexní a rozmanité. Většina zdrojového kódu a programů z této knihy využívá nějakým způsobem událostí. Kapitola 6 a příloha B jsou věnované událostem a jejich interakcím.

JavaScript a CSS

Na základech vybudovaných z DOM a interakcí událostí můžeme stavět dynamické HTML. Ve své podstatě představuje dynamické HTML interakce mezi JavaScriptem a údaji CSS, připojenými k prvkům DOM.

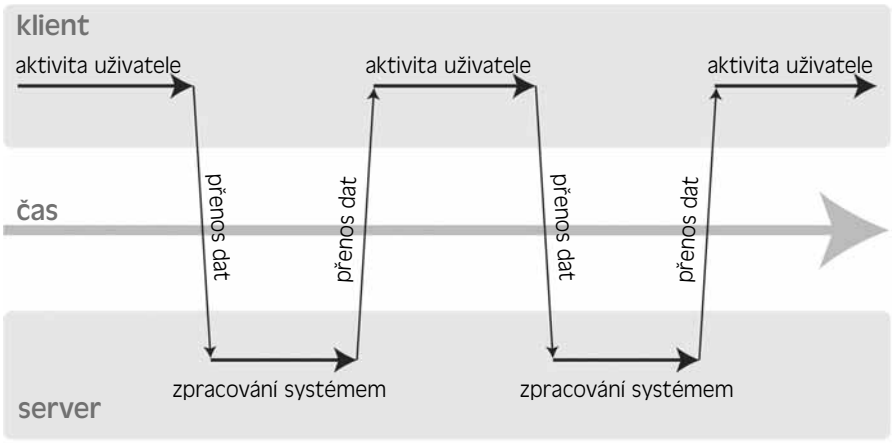
Jazyk CSS (tabulky kaskádových stylů, Cascading Style Sheets) slouží jako standard pro tvorbu jednoduchých, nevtíravých webových stránek, který stále dává programátorům velké možnosti při minimáalizaci nutnosti řešit problémy s kompatibilitou. Základem dynamického HTML je objevování toho, čeho lze dosáhnout komunikací JavaScriptu s CSS a jak této kombinace využít pro tvorbu úžasných výsledků.

Příklady pokročilých interakcí, jako například přesunovatelné prvky a animace, můžeme najít v kapitole 7, kde jsou podrobně popsány.

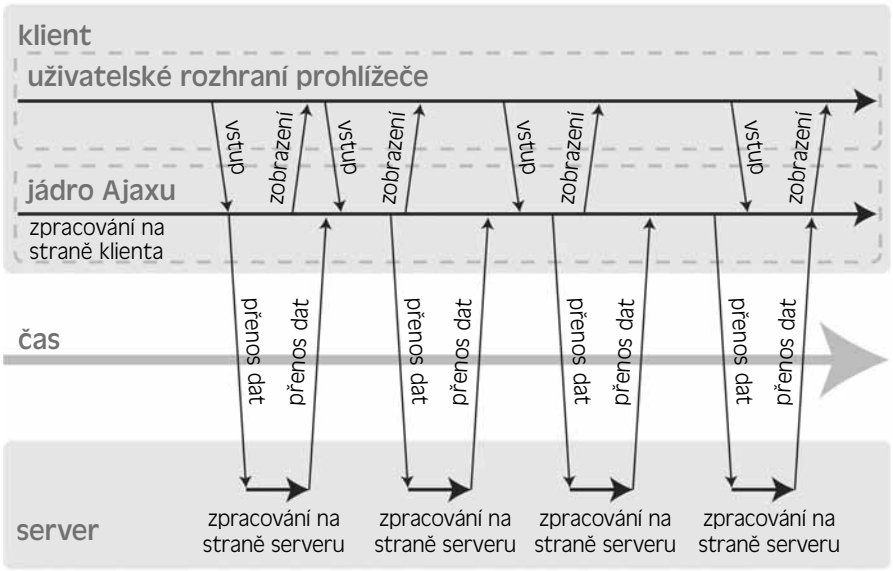
Ajax

Ajax (Asynchornous JavaScript and XML) je termín, který byl poprvé uveden v článku „Ajax: A New Approach to Web Applications" (*http://www.adaptivepath.com/publications/essays/archives/000385.php*) od Jesse James Garretta, spoluzakladatele a prezidenta Adaptive Path, firmy, která se zabývá informační architekturou. Daný článek popisuje

klasický model webové aplikace (synchronní)



model ajaxové webové aplikace (asynchronní)



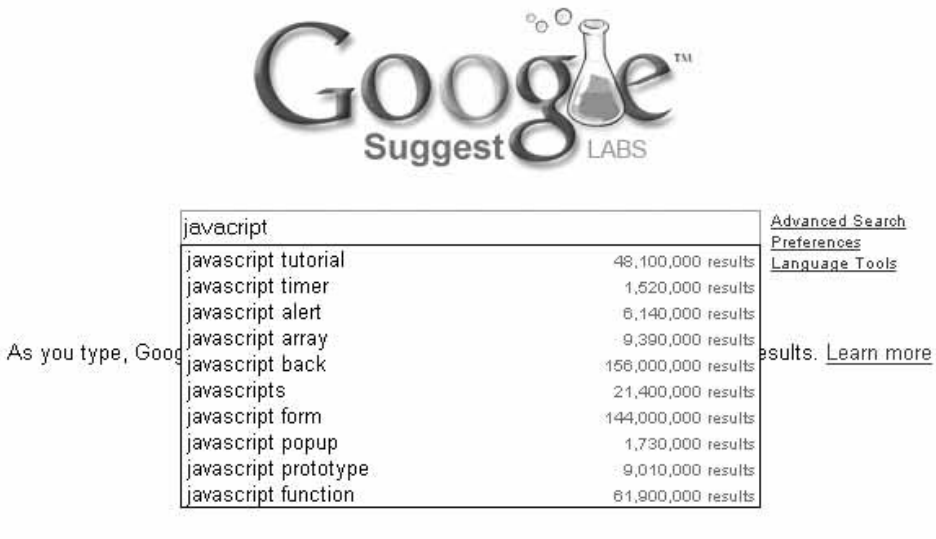
Obrázek 1.3. Diagram z článku „Ajax: A New Approach to Web Applications" ukazuje pokročilou asynchronní interakci mezi klientem a serverem

pokročilé interakce mezi klientem a serverem, když požadujeme nebo nabízíme dodatečné informace.

Termín Ajax zahrnuje tisíce různých datových komunikací, ale všechny mají společný předpoklad – dodatečné požadavky jsou prováděny směrem od klienta k serveru dokonce i v případě, kdy už je stránka kompletně načtena. To umožňuje aplikačním programátorům vytvářet další interakce, aniž by uživatelům zpomalili tradiční běh aplikace. Na obrázku 1.3 vidíme, jak se toky interakcí uvnitř aplikace mění vlivem dodatečných požadavků, které jsou prováděny na pozadí (a většinou bez vědomí uživatele).

Po vydání Garrettova článku vzrostl zájem uživatelů, programátorů, návrhářů a manažerů a s novými aplikacemi, které používají tuto pokročilou úroveň interakce, se doslova „roztrhl pytel". Ironií je, že ačkoliv zájem po vydání tohoto článku vzrostl, technologie, která se skrývá za Ajaxem, je mnohem starší (používá se komerčně už od roku 2000). Hlavním rozdílem však je to, že starší programy komunikovaly se serverem způsobem, který odpovídal určitému webovému prohlížeči (například jen pomocí vlastností prohlížeče Internet Explorer). Jelikož moderní prohlížeče podporují XMLHttpRequest (hlavní metoda pro odesílání a příjem dat XML ze serveru), značně to celou situaci zjednodušilo a umožnilo všem využít těchto výhod.

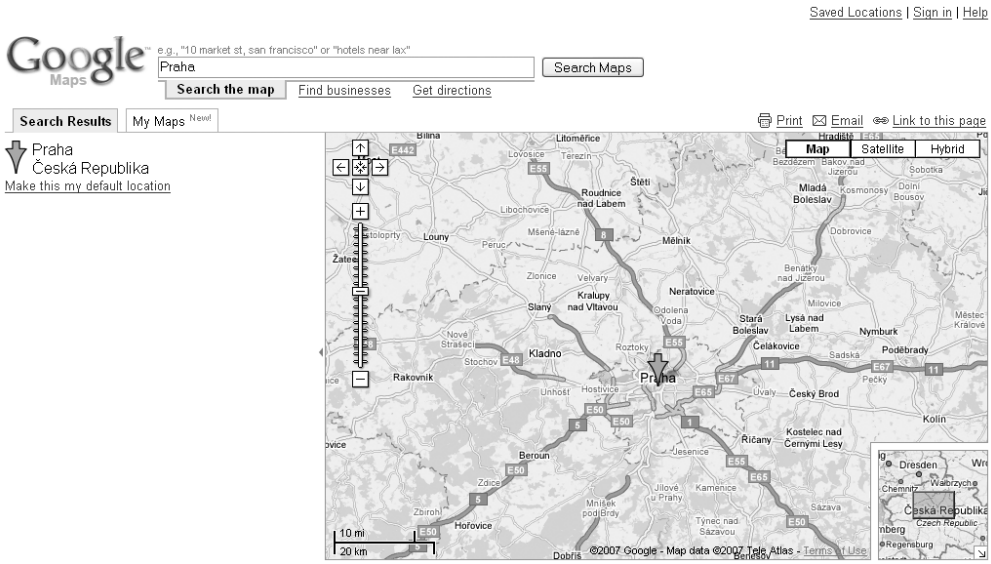
Jestliže nějaká firma někdy byla v čele s tvorbou aplikací pomocí technologie Ajax, pak je to Google. Jednou velmi interaktivní ukázkou, která vyšla těsně před vydáním zmíněného článku o Ajaxu, je Google Suggest. Tato ukázka vám umožní zadat dotaz a již během zadávání vyhledá a navrhne zbytek dotazu, což je vlastnost, které jsme na starých stránkách dosáhnout nemohli. Google Suggest si lze prohlédnout na obrázku 1.4.



Obrázek 1.4. Google Suggest, aplikace dostupná již v době vydání Garrettova článku o Ajaxu, používá asynchronní techniky XML

Další revoluční aplikací od Google je Google Maps, která umožňuje uživatelům pohybovat se po mapě a prohlížet relevantní výsledky, které se zobrazují v reálném čase. Rychlost a použitelnost této aplikace je na rozdíl od podobných aplikací díky použití technik Ajaxu opravdu revoluční. Google Maps najdeme na obrázku 1.5.

Přestože se samotný programovací jazyk JavaScript příliš nezměnil, během posledních několika let jej začali jako plnohodnotný programovací jazyk používat firmy jako Google a Yahoo, což dokazuje, jak se změnilo vnímání tohoto jazyka veřejností a jak vzrostla jeho popularita.



Obrázek 1.5. Google Maps používá řadu technik Ajaxu pro dynamické načítání lokálních informací

Podpora ze strany prohlížečů

Smutnou pravdou programování v JavaScriptu je, že od doby, co je tento jazyk úzce spjat s webovými prohlížeči, které jej implementují a podporují, velmi záleží na tom, jaké webové prohlížeče jsou právě oblíbené. Jelikož uživatelé nemusí nutně používat prohlížeče s nejlepší podporou JavaScriptu, musíme vybrat jen nejdůležitější funkce.

Většina programátorů jednoduše přestala podporovat webové prohlížeče, které způsobovaly spoustu problémů. Je těžké rozhodovat se, zda podporovat webový prohlížeč kvůli šíři jeho uživatelské základny nebo kvůli tomu, že obsahuje naší oblíbenou funkci.

Nedávno firma Yahoo vydala knihovnu v JavaScriptu, pomocí které můžeme rozšířit naše webové aplikace. Spolu s knihovnou firma vydala také pravidla návrhových vzorů pro programátory. Nejdůležitějším dokumentem od Yahoo je zřejmě oficiální seznam prohlížečů, které tuto knihovnu podporují a které nikoliv. Přestože něco podobného

může udělat kdokoliv, jakákoliv společnost, mít k dispozici dokument, který poskytla jedna z nejnavštěvovanějších webových stránek na Internetu, je jistě neocenitelné.

Yahoo zavedl systém stupňů podpory prohlížečů, který každému prohlížeči přiřadí určitý stupeň a podle daného stupně poskytuje prohlížeči jisté funkce. Yahoo rozlišuje tři stupně podpory prohlížečů – A, X a C:

- A-stupňové prohlížeče jsou plně podporované a testované a všechny aplikace od Yahoo v nich zaručeně fungují.
- X-stupňový prohlížeč je podobný jako A-stupňový, Yahoo ví o jeho existenci, ale buď nebyl kompletně testován, nebo se jedná o zcela nový webový prohlížeč. X-stupňovým prohlížečům je nabízena stejná funkčnost jako A-stupňovým a nezbývá než doufat, že si daný prohlížeč s tímto pokročilým obsahem poradí.
- C-stupňové jsou známé jako „špatné" prohlížeče, které nepodporují nezbytné funkce pro běh aplikací od Yahoo. Těmto prohlížečům je nabízen obsah bez JavaScriptu, protože aplikace od Yahoo jsou zcela nevtíravé (proto budou stále fungovat bez přítomnosti JavaScriptu).

Existencí stupňů podpory prohlížečů je ovlivněna i tato kniha. Mohli jsme se již setkat na mnoha místech s termínem moderní prohlížeč, toto označení se shoduje s A-stupňovým prohlížečem podle stupnice od Yahoo. Pokud máme k dispozici konzistentní skupinu funkcí, jistě se nám bude lépe učit a programovat (vyhneme se problémům s nekompatibilitou prohlížečů).

	Win 98	Win 2000	Win XP	Mac 10.0	Mac 10.2	Mac 10.3	Mac 10.3.x	Mac 10.4
IE 7.0	n/a	n/a	A-stupeň	n/a	n/a	n/a	n/a	n/a
IE 6.0	A-stupeň	A-stupeň	A-stupeň	n/a	n/a	n/a	n/a	n/a
IE 5.5	A-stupeň	A-stupeň	n/a	n/a	n/a	n/a	n/a	n/a
IE 5.0	C-stupeň	C-stupeň	n/a	C-stupeň	C-stupeň	C-stupeň	C-stupeň	C-stupeň
Netscape 8.0	X-stupeň	X-stupeň	A-stupeň	n/a	n/a	n/a	n/a	n/a
Firefox 1.5	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň
Firefox 1.0.7	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň	A-stupeň
Mozilla 1.7.12	X-stupeň	X-stupeň	A-stupeň	X-stupeň	X-stupeň	X-stupeň	X-stupeň	X-stupeň
Opera 8.5	X-stupeň	X-stupeň	A-stupeň	C-stupeň	C-stupeň	C-stupeň	X-stupeň	X-stupeň
Safari 1.0	n/a	n/a	n/a	X-stupeň	n/a	n/a	n/a	n/a
Safari 1.1	n/a	n/a	n/a	X-stupeň	X-stupeň	n/a	n/a	n/a
Safari 1.2	n/a	n/a	n/a	X-stupeň	X-stupeň	X-stupeň	n/a	n/a
Safari 1.3	n/a	n/a	n/a	n/a	n/a	X-stupeň	A-stupeň	n/a
Safari 2.0	n/a	n/a	n/a	n/a	n/a	n/a	n/a	A-stupeň

Obrázek 1.6. Tabulka se stupni podpory prohlížečů od Yahoo

Je velmi užitečné přečíst si dokumenty o stupních podpory prohlížečů (lze najít na *http://developer.yahoo.com/yui/articles/gbs/gbs.html*) a tabulku na obrázku 1.6, abychom věděli, čeho chce Yahoo dosáhnout. Tím, že Yahoo rozšiřuje tuto informaci mezi širokou programátorskou veřejnost, buduje neocenitelný standard.

Informaci o tom, které prohlížeče jsou podporovány, najdeme v příloze C této knihy, kde budou popsány všechny výhody a nevýhody jednotlivých webových prohlížečů. Brzy zjistíme, že A-stupňové prohlížeče jsou přesně tím, co potřebujeme, poskytují dostatek funkcí pro programování.

To, jaké prohlížeče chceme podporovat, bude mít vliv na to, jakými funkcemi naše aplikace bude disponovat. Jestliže se třeba rozhodneme podporovat Netscape Navigator 4 nebo Internet Explorer 5, jistě to značně omezí počet funkcí, které můžeme ve své aplikaci použít, protože tyto prohlížeče nepodporují techniky moderního programování.

Když víme, které prohlížeče patří mezi moderní, můžeme využít jejich užitečné funkce, čímž získáme dobrý základ pro další programování. Tímto základem mohou být následující skupiny funkcí:

- Core JavaScript 1.5: Aktuálně nejoblíbenější verze JavaScriptu. Obsahuje všechny funkce pro podporu objektově orientovaného JavaScriptu. Internet Explorer 5.0 nepodporuje zcela úplně verzi 1.5, což je důvod, proč programátoři neradi podporují tento prohlížeč.
- XML Document Object Model (DOM) 2: Standard určený pro dokumenty HTML a XML. Je nezbytný pro programování rychlých aplikací.
- XMLHttpRequest: Základní část Ajaxu – jednoduchá vrstva pro zahajování požadavků HTTP. Všechny prohlížeče tento objekt podporují, s výjimkou Internet Explorer 5.5-6.0, avšak ty podporují zahajování podobným objektem v ActiveX.
- CSS: Nutnost pro navrhování webových stránek, klíčové pro programátory webových aplikací. Všechny moderní prohlížeče podporují CSS, většinou se setkáme s rozpory v prezentaci, které způsobují spoustu problémů. To je také hlavní důvod, proč je Internet Explorer pro Mac tak málo podporován.

Kombinace všech těchto vlastností prohlížečů tvoří základ pro programování webových aplikací v JavaScriptu. Jelikož všechny moderní prohlížeče podporují výše uvedené vlastnosti, máme solidní základ, na kterém budeme stavět po zbytek knihy. Vše, co bude popisovat tato kniha, předpokládá, že prohlížeče, který používáme, podporuje alespoň před chvílí zmíněnou skupinu funkcí.

Shrnutí

Tato kniha se snaží kompletně popsat všechny moderní, profesionální programovací techniky v JavaScriptu, které používají jak běžní programátoři, tak velké firmy a tvoří pomocí nich použitelný, snadno pochopitelný a interaktivní zdrojový kód.

V této kapitole jsme měli možnost zhlédnout přehled všeho, co najdeme v této knize. Podívali jsme se na základy profesionálního programování v JavaScriptu – psaní objek-

tově orientovaného zdrojového kódu, testování zdrojového kódu a rozčleňování do balíků pro distribuci. Dále jsme viděli základy nevtíravého skriptování DOM– krátký úvod do Document Object Model, událostí a interakce mezi JavaScriptem a CSS. Nakonec jsme objevili, co se skrývá za Ajaxem a jak moderní prohlížeče podporují JavaScript. Všechna tato témata jsou dostačující k tomu, aby z nás udělala profesionální programátory v JavaScriptu.